Research Statement

Jana Giceva

My research interests are in systems support for data science and big data to enable the efficient use of modern and future hardware. The scope of my research spans across multiple systems subfields, from the data-processing layer to operating systems, including hardware accelerators for data processing.

Modern system software is in particular challenged by two trends: First, we collect data at unprecedented rates, and making sense out of that data, analyzing it, and managing it is of critical importance to both enterprises and researchers. Apart from designing new models and techniques for analyzing this data deluge, we also need suitable system support to make the analysis more performant, efficient, and applicable in a wider setting. Second, hardware has become more complex and heterogeneous. Instead of boosting CPU frequencies, hardware vendors have increased the number of cores and diversity of computational units within modern machines. As a result, it is now up to the system software to adapt its architecture and implementation in order to achieve better performance and use the potential of increased core count, novel hardware features (*e.g.*, accelerators, system-on-chip), and new technologies (*e.g.*, non-volatile memory).

Addressing the challenges imposed by these trends requires an effort which is beyond what can be typically done within a single layer of the system stack. Therefore, I argue for a *holistic solution* which crosses several layers: understand the requirements of data science workloads, both manage the increasingly diverse set of resources and influence the design of future hardware architectures to better fit the requirements of modern workloads.

My research is based on building real systems and a strong analytical treatment of the performance models related to these systems. In principle, I strive towards high impact research, often in collaboration with other researchers from leading industry labs and academia.

1 Recent and ongoing research

Throughout my PhD studies, I have led and contributed to multiple large-scale system projects at ETH Zurich, Oracle Labs, and Microsoft Research. The focus of my dissertation was on *database/operating system co-design* for running analytical workloads on multicore machines [3–5], but I have also built customized hardware accelerators for data processing [7,9], improved memory management in runtime engines for Big Data systems [6], designed and implemented a rack-scale data processing architecture for hybrid workloads [8], and worked on a scalable relational query execution engine for multicore machines [11].

1.1 Database/Operating system co-design

In my dissertation, I revisited the problem of database/operating system co-design. For decades, database (DB) engines have found the generic interfaces offered by operating systems (OSes) at odds with the need for efficient utilization of hardware resources. This is partly due to the big semantic gap between the two layers. The rigid DB/OS interface does not allow for knowledge to flow between them, and as a result: (1) the operating system is unaware of the requirements of the database, and provides a set of general purpose policies and mechanisms for all applications running on top, (2) the database can, at best, duplicate a lot of the OS functionality internally at the cost of absorbing significant portion of additional complexity in order to efficiently use the underlying hardware – an approach that does not scale with the current pace of hardware developments.

My work approaches this problem from two perspectives: (a) by reducing the knowledge gap between the DB and OS, and (b) by architecting the OS to allow dynamic splitting of the machine resources into a *control* and *compute* plane, where the compute plane kernels can be specifically tailored to the needs of data processing applications.

More specifically, in COD [4] I introduced an *OS policy engine* which can reason about system state, contains a knowledge base with a model of the machine and its resources, and understands applications' requirements. Via a new *declarative* interface the database can now push some of its knowledge to the OS policy engine, where the system state resides. For instance, the database can share cost models of its operators (*e.g.*, a relational scan) with

the OS. The policy engine can then use them derive how the operator's performance is affected by its resource allocation decisions. In follow up work [3], I have shown how using the data dependency graph of a DB query plan, the resource footprint of each relational operator in the graph, and the system state from the OS policy engine, we compute an optimal operator-to-core deployment. In the experimental evaluation I showed that it minimizes resource utilization (up to 7x more efficient for the TPC-W workload benchmark) without sacrificing tail latency and throughput predictability.

In Basslet [5] I propose using the *multi-kernel* OS architecture [1] to split the machine resources into a control and compute plane. The control plane runs the full OS kernel/stack and the main application threads. The compute plane, however, is a lightweight kernel (LWK) which is solely responsible for executing jobs on behalf of applications in a noise-free environment, which can be customized to the requirements of specific workloads. The current prototype is based on the Barrelfish OS [12] and extends the traditional UNIX process model to also offer task-based execution for parallel data-intensive jobs on hardware *isolation* islands via a kernel-integrated runtime scheduler. The system provides an interface that captures the different parallelism job requirements from complex multithreaded applications, analogous to MapReduce cluster type frameworks.

1.2 Architecture for hybrid data processing

Many businesses today rely on data driven decisions, preferably by doing analytics on fresh data. As a result, it is increasingly important that modern data processing systems provide support for efficient execution of hybrid workloads applications. Even though there are several enterprise and research systems which offer support to run such HTAP workloads, concurrent execution significantly degrades the throughput of both workloads.

The system architecture we proposed and implemented (BatchDB [8]) achieves good performance, provides a high level of data freshness, and minimizes the load interaction between the transactional and analytical engines, thereby allowing for real time analysis over fresh data under tight service level objectives for throughput and latency for both type of workloads. To achieve these goals, BatchDB's architecture relies on primary-secondary replication with dedicated replicas optimized for a particular workload type, a light-weight propagation mechanism for transactional updates, and isolated resource allocation either by scheduling the different engines on separate NUMA-islands or across different machines. This project has already attracted a lot of attention from leading industry companies such as Oracle Labs and SAP.

1.3 Hardware accelerators for data processing

Almost a decade ago as CPU frequencies have stopped increasing, multicore systems became prevalent. As a result, the effort of improving the system and application performance has transferred to the developer. This becomes more difficult with trends of increasing hardware heterogeneity and various new hardware accelerators optimized for different purposes (*e.g.*, Google's TPU for machine learning, NVIDIA's Tesla for deep neural networks, Oracle's SPARC M7 for advanced relational analytics). As opposed to building system software which continuously tries to keep up with the pace at which hardware evolves, we have taken the complementary approach and also explored the potential of hardware/software co-design and directly influencing the upcoming hardware features.

In this direction, I contributed to the early design and development of Oracle's RAPID accelerator [9]. More concretely, I was responsible for the implementation of a (distributed) join operator co-designed with the new hardware architecture, and building a detailed cost model. I have also worked on projects which explore hybrid platform co-processing (CPU+FPGA) for relational operations. In a recent paper we demonstrated the feasibility of the approach, by implementing a hash-based partitioner on an FPGA circuit [7], which can be easily integrated as a pre-phase for highly optimized operations (*e.g.*, a radix hash join).

2 Future research

It is an exciting time to be a systems researcher. Looking forward, I want to re-design the system stack support for data science and modern data analytics workloads to take advantage of the many changes taking place in hardware architecture and computing platforms. I discuss some concrete ideas in this section, but I would be happy to continue my work on a wider range of topics and in collaboration with colleagues from adjacent areas.

2.1 Support for Data Science workloads

Sub-operators as first class citizens. Databases have over decades developed a wealth of technologies that have been successfully used in many settings. Unfortunately, most of it is left aside in the pursuit of performance (*e.g.*, NoSQL) or new functionality (*e.g.*, graph processing, machine learning). Part of this is due to the coarse granularity of the principal DB components – the relational operators. Thus, my proposal is to make databases a more flexible platform with extended support for new data types and operations by lowering the level of abstraction: *instead of working with SQL, the DB runtime engine should compile, optimize, and run a collection of sub-operators for manipulating and managing data*. As a result, the database will shift from being a virtual machine executing SQL programs to a language runtime executing a complex instruction set architecture (ISA) made up of sub-operators. Such a change enables three things: (1) building more expressive dataflows for a variety of data scientific workloads (*e.g.*, machine learning, data mining, etc.), (2) easier portability across different platforms, and (3) influencing the design of future hardware features. Our FPGA-based partitioner is just one example of such a sub-operator. Its circuit could (in the future) be integrated on-chip, and be used as a building component for many different dataflows beyond SQL hash-join and aggregation.

Adaptive OS for data science. The policies and mechanisms suitable for data science workloads can vary. A transactional workload is characterized by latency sensitive, short running jobs, interacting with both the storage and network layers. Analytical queries are more bandwidth intensive, rely on main memory processing and are typically longer running parallel jobs. Thus, an important property of the Basslet OS architecture from my dissertation work is its adaptability. It allows customizing the LWKs to the different workloads requirements, and dynamically updating the compute plane as the workload mix changes. As a follow up project I intend to use our hybrid data processing engine (BatchDB) on top of the Basslet OS architecture, and implement the suitable compute plane kernels for both workload types. That way we can evaluate the benefits of a customized system stack for hybrid workloads.

2.2 Future rack-scale machines

New hardware technologies started changing the landscape of data center computing, and rack-scale computers (with high-bandwidth low-latency network fabrics connecting thousands of cores and terabytes of memory) are projected to become the dominant unit of deployment in many large scale companies. From a systems perspective this opens up plethora of questions, including *how to co-design the multiple layers of the software stack in order to improve both performance and efficiency*. I plan to build upon my current research to address the challenges of heterogeneous resources, managing memory, storage, and network in the light of such new technologies (*e.g.*, NVRAM, active DRAM, system- or network-on-chip, etc.).

Heterogeneous hardware. From the OS perspective an interesting challenge is how to capture and model the diversity of resources, monitor their utilization, and expose some of that information to the applications. But also to explore how the OS policy engine can assist data processing applications to map their work units onto a diverse set of platforms (*e.g.*, a Xeon Phi or an FPGA). Furthermore, current trends suggest that we are headed towards *active* resources (*e.g.*, smart NICs, active near-memory processing, intelligent storage, etc.). As oppose to treating them as external devices, we should have the OS manage their computational capacities (*i.e.*, the control plane) and export their services directly to the applications (*i.e.*, the data or compute plane). By design the Basslet architecture enables us to customize the compute plane kernels for the underlying platform, and scale to large rack-scale machines.

More efficient use and access to I/O devices. Due to lack of support from conventional operating systems, most data processing engines today optimize their implementations for network operations by removing the OS from the critical data path (*e.g.*, using RDMA over InfiniBand) and rely on the network interface card (NIC) driver to do the resource multiplexing. Recent research OSes [2, 10] use hardware virtualization capabilities of I/O devices to give applications direct access to the device without involving the OS kernel – data plane. I would like to explore: (1) how the policies devised by the control plane (*i.e.*, the OS) can be improved given more knowledge of application's needs; and (2) how to design and implement the I/O system stack so that performance improves without entirely offloading the complexity to the application.

Memory management. For certain classes of applications and workloads, the virtual memory system is becoming the bottleneck (*e.g.*, overhead of virtual to physical memory translation, managing terabytes of main memory, etc.). Recently proposed techniques which allow applications to take over more control over their address space and virtual to physical page translation (*e.g.*, self-paging, direct segments, etc.) could be of use to applications like databases, especially at rack-scale.

References

[1] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 29–44, 2009.

[2] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 49–65, 2014.

[3] J. Giceva, G. Alonso, T. Roscoe, and T. Harris. Deployment of query plans on multicores. *Proc. VLDB Endow.*, 8(3):233–244, Nov. 2014.

[4] J. Giceva, T.-I. Salomie, A. Schüpbach, G. Alonso, and T. Roscoe. COD: Database/Operating System Co-Design. In *CIDR '13*.

[5] J. Giceva, G. Zellweger, G. Alonso, and T. Rosco. Customized OS Support for Data-processing. In *Proceedings of the 12th International Workshop on Data Management on New Hardware*, DaMoN '16, pages 2:1–2:6, 2016.

[6] I. Gog, J. Giceva, M. Schwarzkopf, K. Vaswani, D. Vytiniotis, G. Ramalingan, D. Murray, S. Hand, and M. Isard. Broom: Sweeping out Garbage Collection from Big Data Systems. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, 2015.

[7] K. Kara, J. Giceva, and G. Alonso. FPGA Based Data Partitioning. In *To Appear in SIGMOD '17*, 2017.

[8] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *To Appear in SIGMOD '17*, 2017.

[9] Oracle Labs. Project RAPID. https://labs.oracle.com/pls/apex/f?p=labs:49:::::P49_PROJECT_ID:14, November 2016.

[10] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The Operating System is the Control Plane. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 1–16, 2014.

[11] T.-I. Salomie, I. E. Subasu, J. Giceva, and G. Alonso. Database Engines on Multicores, Why Parallelize when You Can Distribute? In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 17–30, 2011.

[12] The Barrelfish Project. Barrelfish Operating System. www.barrelfish.org, accessed 2016-08-12.